گزارش ششم، دستور آزمایش پنجم

شهزاد ممیز ۸۱۰۱۰۰۲۷۲

محمد امانلو ۸۱۰۱۰۰۰۸۴

آشنایی با برخی از دستورات Unix

در اصل زمانی که صحبت از UNIX به میان می‌آید، منظور یک سیستم عامل متن باز است که در دهه‌ی ۱۹۷۰ توسط کارکنان ATT Bell Labs توسعه یافت. UNIX برای انعطاف‌پذیری و چندکاربره بودن شناخته شده است و از آنجا که مبنای خیلی از سیستم‌های عامل دیگر مثل Linux و BSD قرار گرفته است، تاثیر قابل توجهی در صنعت IT داشته است.
دستورات UNIX بخش مهمی از کار با این سیستم عامل هستند. در ادامه به شرح برخی از پرکاربردترین دستورات پرداخته شده است:

۱. ` :`ls لیست کردن فایل‌ها و پوشه‌ها
این دستور برای نمایش فایل‌ها و پوشه‌های موجود در مسیر فعلی استفاده می‌شود.

۲. ` :`cd [directory]تغییر پوشه (Change Directory)
این دستور برای جابجایی بین پوشه‌ها به کار می‌رود. ` ]`directory]جایگزین نام پوشه‌ای می‌شود که می‌خواهید به آن بروید.

۳. ` :`mkdir [directory name] ایجاد یک پوشه جدید (MaKe DIRectory)
از این برای ایجاد یک پوشه جدید استفاده می‌شود.

۴. ` :`rmdir [directory name]حذف یک پوشه (ReMove DIRectory)
این دستور برای حذف پوشه‌ها به کار می‌رود. توجه داشته باشید که پوشه باید خالی باشد.

۵. ` :`cp [source] [destination] کپی کردن فایل‌ها و پوشه‌ها (CoPy)
برای کپی کردن فایل‌ها و پوشه‌ها از محلی به محل دیگر استفاده می‌شود.

۶. ` :`mv [source] [destination] جابجایی یا تغییر نام فایل‌ها (MoVe)

جابجایی فایل‌ها و پوشه‌ها یا تغییر نام آن‌ها.

۷. `rm [file name]` : حذف فایل‌ها (ReMove)
یک فایل یا چندین فایل را حذف می‌کند.

۸. `chmod [options] [permissions] [file name]` : تعیین مجوزهای فایل (CHange MODe)
برای تغییر دادن مجوزهای یک فایل یا پوشه به کار می‌رود.

۹. `grep [search_pattern] [file]` : جستجوی متن
برای جستجوی الگوها داخل فایل‌ها استفاده می‌شود.

۱۰. `pipe ()` : چرخاندن خروجی یک دستور به دستور دیگر
برای ترکیب چند دستور و استفاده از خروجی یک دستور به عنوان ورودی دستور بعدی.

از ویژگی‌های UNIX استفاده از پایپ‌لاین‌ها () و ریدایرکت‌ها (<, >) است که امکان مدیریت پیچیده‌ی ورودی و خروجی‌ها را فراهم می‌سازد.

این فقط چند نمونه از دستورات UNIX است. سیستم عامل‌های مبتنی بر UNIX مانند Linux دارای چندین دستور دیگر نیز هستند که فراتر از این‌ها برای عملیات مختلفی که بر روی فایل‌ها و پردازش‌ها انجام می‌شود، استفاده می‌شوند.

معرفی برخی از اصطلاحات کاربردی

۱. Regex:

`regex` کوتاه‌شده‌ی Regular Expression (عبارت منظم) است. یک regex یک الگوی متنی که برای جستجو، بازشناسی یا جایگزینی متون در یک رشته کاراکتر استفاده می‌شود. با استفاده از سینتکس خاصی، regex قادر است پیچیده‌ترین الگوهای متنی را ورای جستجوی ساده در متون به دست آورد و آن‌ها را منطبق سازی کند.

اجزای اصلی:

- کاراکترهای ساده: هر کاراکتر ساده‌ای مثل a یا `۱` یک الگوی regex است که دقیقاً با خود آن کاراکتر مطابقت دارد.
- نقطه: نمایانگر هر کاراکتری (به جز پایان خط یا newline) است.
- کاراکتر کلاس‌ها : عملیات مطابقت با یک مجموعه انتخابی از کاراکترها را امکان‌پذیر می‌سازد. مثال: [abc]با a، b، یا c مطابقت پیدا می‌کند.
- کوانتیفایرها `* + ? {}`: تعداد دفعات تکرار یک کاراکتر، گروه یا کلاس کاراکتری را تعیین می‌کنند. مثلا a* با هیچ a یا چندین a متوالی مطابقت دارد.
- گروه بندی `()`: برای تعیین ترتیب عملیات یا جداسازی بخش‌های متن برای بازیافت بعدی.
- انکرها `^ $`: `^` ابتدای یک رشته یا یک خط را نشان می‌دهد و `$` پایان یک رشته یا خط را.
- فرار کاراکتر `\`: برای جستجوی کاراکترهای خاصی که معانی ویژه‌ای در regex دارند استفاده می‌شود.

کاربردها:

regex می‌تواند در بسیاری از زمینه‌ها از جمله برنامه نویسی، ویرایش متون، مدیریت دیتابیس و تحلیل دیتا به کار رود:

- جستجو و جایگزینی متن: در ویرایشگرهای متن و ابزارهای توسعه مانند sed و awk.
- اعتبارسنجی: تایید فرمت‌های ورودی مانند ایمیل‌ها، شماره‌های تلفن و آدرس‌ها در فرم‌های وب.
- پارس کردن داده‌ها: استخراج اطلاعات مشخص از متن‌های بزرگ و داده‌های ساخت‌یافته مانند فایل‌های XML یا CSV.

regex ابزار قدرتمندی است اما می‌تواند بسیار پیچیده باشد و گاهی سنگین شود، پس استفاده از آن همیشه باید همراه با در نظر گرفتن هزینه‌های عملکردی و خوانایی کد باشد.

۲. AWK

AWK یک زبان برنامه‌نویسی مبتنی بر الگو است که اصالتا برای پردازش داده‌ها و گزارش‌سازی طراحی شده است. این زبان به نام‌های سه مخترع آن، آهو، واینبرگر و کرنیگن، نامیده شده است.

AWK به خصوص برای کار با داده‌های متنی که در فرمت رکوردها و فیلدها سازماندهی شده‌اند (به عنوان مثال فایل‌های CSV بسیار مفید است. اگر پرونده‌ای شامل ستون‌ها و ردیف‌ها باشد، AWK می‌تواند برای انجام عملیات‌های خاص روی هر ستون یا سطر استفاده شود. در هر خط از یک فایل، فیلدهایی که توسط جداکننده‌ها (مانند فاصله یا کاما) جدا شده‌اند، به طور خودکار تشخیص داده شده و توسط AWK پردازش می‌شوند.

برنامه‌نویسی در AWK معمولاً شامل نوشتن یک اسکریپت کوچک، که می‌توان آن را مستقیما در خط فرمان نوشت یا در یک فایل جداگانه قرار داد. این اسکریپت‌ها معمولاً از الگوها برای تعریف فرایندهایی که باید روی داده‌ها اجرا شوند و اکشن‌ها استفاده می‌کنند که مشخص می‌کنند چه چیزی باید انجام شود.

AWK امکانات پیشرفته‌تری نظیر متغیرها، عملگرهای ریاضی، توابع ساخته شده و حتی توابع تعریف شده توسط کاربر را نیز در خود جای داده است. بنابراین، زبانی قدرتمند برای پردازش و تحلیل داده‌های متنی به حساب می‌آید.

۳. SED

Sed که مخفف Stream Editor است، یک ابزار متنی در سیستم عامل‌های یونیکس‌مانند است که برای پردازش متن خودکار، به کار بردن تغییرات در فایل‌ها، و استخراج اجزا مشخصی از فایل‌های متنی به کار می‌رود. Sed برای پیاده‌سازی تغییرات متنی ساده و پیچیده به شکل غیرتعاملی استفاده می‌شود، به این معنا که می‌تواند به صورت خودکار بدون نیاز به ویرایش دستی متن توسط کاربر عمل کند.

توانایی‌های SED شامل افزودن، حذف، جستجو و جایگزینی متون است. این ابزار اغلب در اسکریپت‌های شل و در فرآیندهایی که نیاز به اصلاحات پرونده‌های متنی در خط فرمان دارند، به کار می‌رود.

چگونگی کار:

SED اساساً خروجی را از یک منبع (معمولاً فایل یا ورودی استاندارد) می‌خواند، و آن را بر اساس دستورالعمل‌های برنامه‌نویسی شده در یک اسکریپت SED یا به صورت مستقیم از خط فرمان پردازش می‌کند. نتیجه معمولاً به خروجی استاندارد (صفحه نمایش) فرستاده می‌شود.

اسکریپت های درخواست شده در صورت پروژه

در این بخش دستوراتی که در صورت پروژه جهت پیاده سازی و انجام ذکر شده است نوشته می‌شود.

بخش اول. پیاده سازی کد دریافت n امین عدد دنباله فیبوناچی:

سری فیبوناچی یک دنباله عددی است که هر عدد آن معادل جمع دو عدد قبل از خود می‌باشد. این دنباله از دو عدد ۰ و ۱ شروع می‌شود، اگرچه در بعضی تعاریف به جای ۰ و ۱، با دو عدد ۱ شروع می‌شود. این سری به این صورت پیش می‌رود:

۰, ۱, ۱, ۲, ۳, ۵, ۸, ۱۳, ۲۱, ۳۴, ...

همانطور که مشاهده می‌کنید، پس از دو اولین عدد، هر عدد نتیجه جمع دو عدد قبلی است:

- ۰ (می‌توان آن را در نظر نگرفت)
- ۱ (ابتدایی)
- ۱ (ابتدایی) = ۰ + ۱
- ۲ = ۱ + ۱
- ۳ = ۱ + ۲
- ۵ = ۲ + ۳
- ۸ = ۳ + ۵
- ۱۳ = ۵ + ۸
- و به همین ترتیب...

سری فیبوناچی اهمیت ویژه‌ای در علوم کامپیوتر و ریاضیات دارد و در طبیعت نیز دیده می‌شود، به طوری که الگوهای مرتبط با آن می‌توانند در مواردی مانند توزیع برگ‌ها، شکل‌گیری کهکشان‌ها، الگوهای جمعیتی برخی از موجودات زنده، و حتی دینامیک‌های اقتصادی و بازار سهام مشاهده شوند.

در ذیل اسکریپتی برای محاسبه آن نوشته شده است.

```
  GNU nano 4.8
#!/bin/bash

# Read the input from the user
echo -n "Enter the value of n to compute the nth Fibonacci number: "
read n

# Initialization of the first two Fibonacci numbers
a=0
b=1

# Check if the input number is less than 0
if [ $n -lt 0 ]; then
    echo "Please enter a non-negative integer."
    exit 1
fi

# Compute the Fibonacci number
for (( i=0; i<n; i++ )); do
    fib=$a
    next=$((a + b))
    a=$b
    b=$next
done

echo "The $n-th Fibonacci number is: $fib"

exit 0
```

بررسی صحت:

```
mohammad@mohammad-VirtualBox:~$ bash CW5.sh
Enter the value of n to compute the nth Fibonacci number: 10
The 10-th Fibonacci number is: 34
mohammad@mohammad-VirtualBox:~$
```

بخش دوم. در این بخش به پیاده سازی اسکریپت insertion sort می‌پردازیم و در ادامه یک سری از کلمات را از کاربر دریافت کرده و آن ها را مطابق حروف الفبا مرتب سازی می‌کنیم.

insertion sort یک الگوریتم مرتب سازی ساده است که به شیوه ای کار می کند که شما کارت های بازی را در دست خود مرتب می کنید. آرایه به طور مجازی به دو بخش مرتب و نامرتب تقسیم می شود .مقادیر از بخش نامرتب انتخاب شده و در مکان صحیح در بخش مرتب قرار می گیرند. در هر تکرار، الگوریتم insertion sort یک عنصر را از داده های ورودی حذف می کند، مکانی که آن عنصر در لیست مرتب می باشد را پیدا می کند، و آن را در آنجا قرار می دهداین الگوریتم برای مجموعه داده های کوچک بسیار کارآمد است

در ذیل این اسکریپت پیاده سازی شده است.

```
GNU nano 4.8                                                        insertion.
#!/bin/bash

# Function for insertion sort
insertion_sort() {
    arr=("$@")
    len=${#arr[@]}

    for ((i = 1; i < len; i++)); do
        key=${arr[$i]}
        j=$((i - 1))

        # Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current position
        while [[ $j -ge 0 && ${arr[$j]} > $key ]]; do
            arr[$((j + 1))]=${arr[$j]}
            j=$((j - 1))
        done
        arr[$((j + 1))]=$key
    done

    echo "Sorted words: ${arr[*]}"
}

# Read words from user
echo "Enter words separated by space:"
read -a words

# Call insertion sort function
insertion_sort "${words[@]}"
```

بررسی صحت کد:

```
mohammad@mohammad-VirtualBox:~$ bash insertion.sh
Enter words separated by space:
Hello my name is Mohammad
Sorted words: Hello is Mohammad my name
mohammad@mohammad-VirtualBox:~$
```

بخش سوم. اسکرپتی بنویسید که تشخیص دهد رشته ورودی نشانگر یک آدرس IP صحیح است یا خیر؟

در ذیل نحوه پیاده سازی این اسکریپت ذکر شده است.

```
  GNU nano 4.8
# Function to validate IP
validate_ip() {
    ip=$1
    IFS='.' read -ra addr <<< "$ip"
    if [ ${#addr[@]} -eq 4 ]
    then
        for i in "${addr[@]}"
        do
            if ! [[ $i =~ ^[0-9]+$ ]] || [[ $i -lt 0 ]] || [[ $i -gt 255 ]]
            then
                echo "Invalid IP address"
                return
            fi
        done
        echo "Valid IP address"
    else
        echo "Invalid IP address"
    fi
}

# Read IP from user
echo "Enter IP address:"
read ip

# Call validate IP function
validate_ip $ip
```

در ذیل نحوه عملکرد این اسکریپت بررسی شده است.

```
mohammad@mohammad-VirtualBox:~$ bash IP.sh
Enter IP address:
123
Invalid IP address
mohammad@mohammad-VirtualBox:~$ bash IP.sh
Enter IP address:
172.23.23.11
Valid IP address
mohammad@mohammad-VirtualBox:~$
```

بخش چهارم. در این بخش باید فایل ها را به صورت درختی نمایش دهیم.

در این قسمت از یک نسخه مجازی رزبری استفاده شده. در این نسخه کامند tree و پکیج منیجر apt نصب نشده اند. با روش زیر این دو را نصب می کنیم.

```
pi@raspberrypi ~/session06 $ tree
-bash: tree: command not found
pi@raspberrypi ~/session06 $ sudo apt install tree
```

سپس اسکریپت زیر را پیاده سازی می‌کنیم.

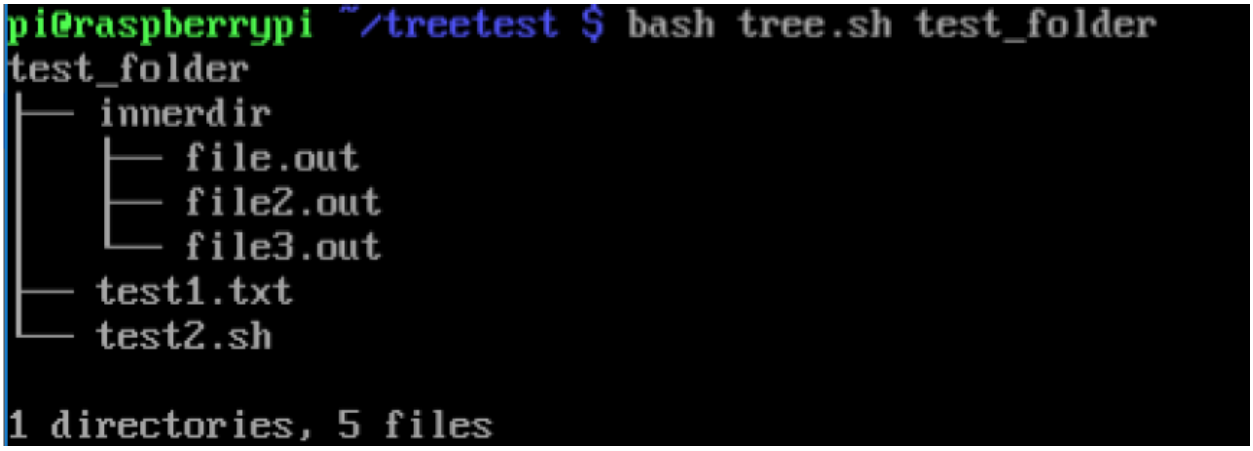
```
#!/bin/bash

if [[ ! -n "$1" ]]; then
  echo "usage: $0 <path>"
  exit 1
fi

if [[ ! -d "$1" ]]; then
  echo "Path is not a folder."
  exit 2
fi

tree "$1"
```

در نهایت خروجی مشابه زیر خواهد بود.


pi@raspberrypi ~/treetest $ bash tree.sh test_folder
test_folder
├── innerdir
│ ├── file.out
│ ├── file2.out
│ └── file3.out
├── test1.txt
└── test2.sh
1 directories, 5 files